University of California, Los Angeles
School of Engineering and Applied Science
Department of Electrical and Computer Engineering
ECE 132A – Introduction to Communication Systems
Professor Flavio Lorenzelli
TA: Manie Tadayon

Soft Demodulation Using Neural Networks

Joani Bajlozi – 804981541
Sanjana Sarda - 104940016

Abstract

On almost every modern-day receiver, soft demodulation of received symbols is used to turn the symbols back into the original conveyed soft bits, or bit log likelihood ratios (LLRs). This project incorporated the use of a neural network to predict log likelihood ratios from a sequence of conveyed bits, based on five common modulation techniques, which included different types of Quadrature Amplitude Modulations and Phase Shift Keying. The project found that for these modulations, the use of a neural network produces very accurate LLR estimates with an order of magnitude less operations than using the direct exact LLR calculation, which shows the potential usefulness in using machine learning to optimize the existing physical-layer design of communication systems.

I. INTRODUCTION

In every current modern day receiver, two crucial processes are demapping received symbols back into their embedded soft bits and then passing them through error correction coding. The soft bits are expressed in terms of their log-likelihood ratios, which give the receiver a certain level of confidence on what each soft bit may be. In theory, the likelihood ratio is defined as the probability of the received bit being one divided by the probability of the received bit to be zero. The log of this ratio is taken in order to basically amplify the results and give the receiver a larger space in order to make its judgement, which makes the process much clearer.

To evaluate LLRs exactly, one must use the log-MAP scheme, which entails computing the logarithm of the ratio between the maximum a-posteriori (MAP) probabilities of the bit's two hypothesis based on the observed symbol. However, this approach is impractical in realistic systems as the computational complexity greatly expands as the size of the symbol constellation used gets larger, which means that it is simply impractical to use the exact method. Instead, modern receivers may use a popular approximation called the max-log-MAP algorithm, which eliminates the need to compute complex exponential and logarithmic functions, but nevertheless, still uses an extensive number of operations that still scale in size based on the constellation size

An alternative method to computing LLRs could be to train a neural network to recognize and classify LLRs based on the received symbols, which would greatly reduce the strain on the receiver, reducing costs and increasing efficiency. In this project, the LLRs were approximated for several modulation schemes, including Gray-coded quadrature amplitude modulation (QAM) and Gray-coded phase shift keying (PSK), which were then used to train and test a neural network.

II. THEORY

LLR Computation for BPSK:

The received signal is given by the expression below:

$$y = c_i + n \qquad i = 0,1$$

$$n \sim N(0, \sigma^2)$$

For BPSK (assuming unit energy), the received signal has the following distribution:

$$y \sim N(1, \sigma^2) \quad if\ c_0\ is\ transmitted$$

$$y \sim N(-1, \sigma^2) \quad if\ c_1\ is\ transmitted$$

The log-likelihood ratio of the transmitted bit is then given by:

$$LLR = \ln\left(\frac{P(b = 1|\, r)}{P(b = -1|\, r)}\right)$$

Using Bayes' Rule, we can obtain the following expression:

$$LLR = \ln\left(\frac{P(y|\, b = 1)}{P(y|\, b = -1)}\frac{P(b = 1)}{P(b = -1)}\right)$$

$$= \ln\left(\frac{P(y|\, b = 1)}{P(y|\, x = -1)}\right) + \ln\left(\frac{P(y = 1)}{P(y = -1)}\right)$$

Since our symbol generation in MATLAB had an equal probability of generating a 1 or -1, the second term becomes $\ln 1$, which is equal to zero. Since our implementation on MATLAB placed the symbols on the imaginary axis instead of the real axis, we will be using y_{im} instead of y_{re} to compute the LLRs. We are then just left with the following expression:

$$LLR = \ln\left(\frac{P(y|\, b = 1)}{P(y|\, b = -1)}\right) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2}}\right)$$

If we simplify, we can obtain the following expression:

$$LLR = -\frac{1}{2}\left(\frac{y_{im} - 1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{im} + 1}{\sigma}\right)^2 = \frac{1}{2}\left(\frac{y_{im}^2 + 2y_{im} + 1 - y_{im}^2 + 2y_{im} - 1}{\sigma^2}\right)$$

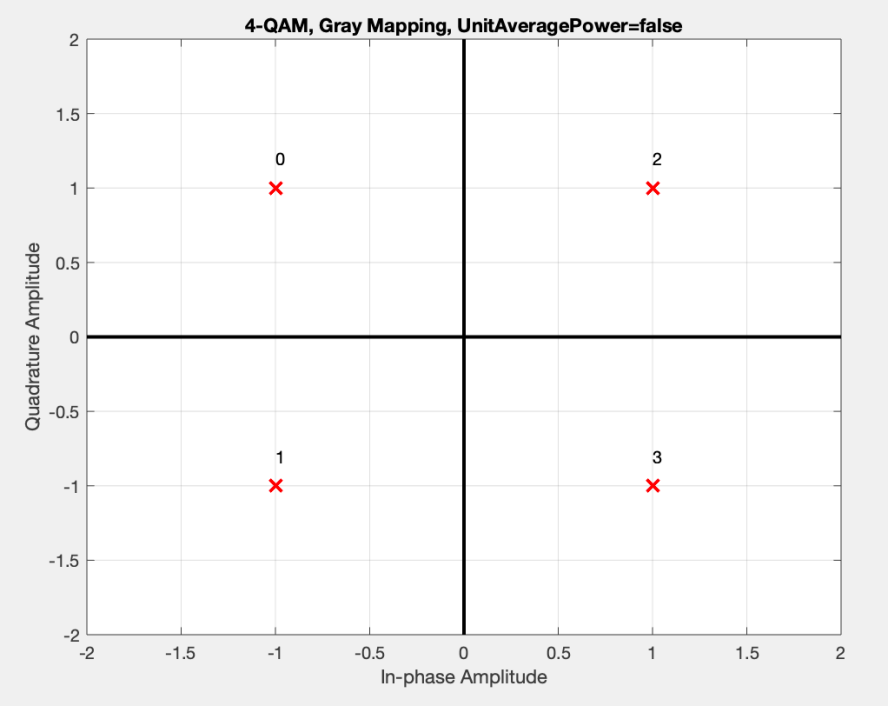
$$LLR = \frac{2}{\sigma^2}y_{im}$$

where y_{im} is the imaginary part of the received symbol.

Each of the circled expressions was used to compute the LLRs on MATLAB, that were in turn used to train the neural network.

LLR Computation for 4QAM:

This form of modulation transmits 4 bits, which means that we will need to compute four different LLR's. The gray coding scheme used by the `qammod` function in MATLAB is shown below:



The numbers shown in the figure above are converted into 4-bit numbers, which correspond to 4 different LLRs, as shown in the chart below:

$Decimal\ Number$	$b_0 b_1 b_2 b_3$
0	00
1	01
2	10
3	11

Now, we are ready to calculate the LLRs for each bit. First, the received signal comes in the form:

$$y = c + n$$

$$n \sim N(0, \sigma^2)$$

$$c \in \{\pm 1 \pm j$$

First, we start off with the first bit, b_0, where we can see from the gray mapping that it will be 0 when the real part of the symbol is negative and it will be 1 when the real part of the symbol is positive. Therefore, we can clearly see that the theoretical LLR for this bit will be defined as shown below:

$$LLR_{b0} = \ln\left(\frac{P(y|\, b_0 = 1)}{P(y|\, b_0 = 0)}\right) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2}}\right)$$

$$LLR_{b0} = -\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2$$

$$LLR_{b0} = \frac{2}{\sigma^2} y_{re}$$

Now, for the second bit, b_1, we can see from the gray mapping that it will be 0 when the imaginary part of the received symbol is positive, and it will be 1 when the imaginary part of the received symbol is negative. Therefore, we can clearly see that the theoretical LLR for this bit will be defined as shown below:

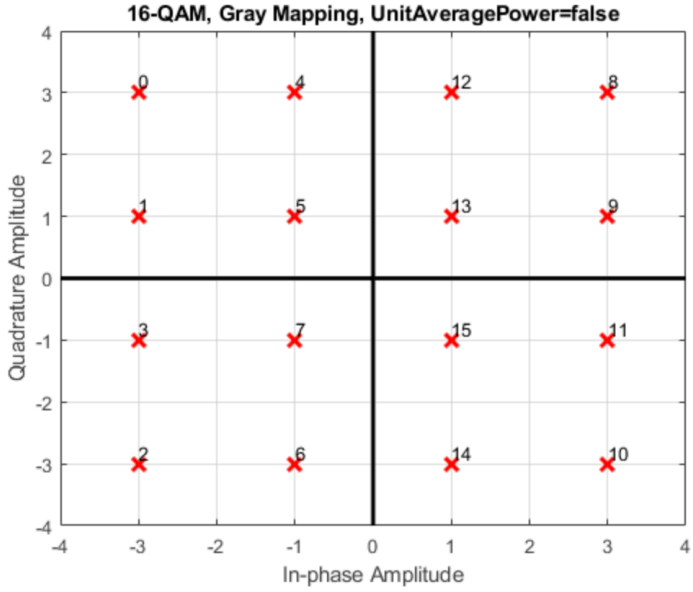
$$LLR_{b1} = \ln\left(\frac{P(y|\, b_1 = 1)}{P(y|\, b_1 = 0)}\right) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2}}\right)$$

$$LLR_{b1} = -\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2$$

$$LLR_{b1} = -\frac{2}{\sigma^2} y_{im}$$

LLR Computation for 16QAM:

This form of modulation transmits 4 bits, which means that we will need to compute four different LLR's. The gray coding scheme used by the `qammod` function in MATLAB is shown below:



The numbers shown in the figure above are converted into 4 bit numbers, which correspond to 4 different LLRs, as shown in the chart below:

$Decimal\ Number$	$b_0b_1b_2b_3$
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Now, we are ready to calculate the LLRs for each bit. First, the received signal comes in the form:

$$y = c + n$$

$$n \sim N(0, \sigma^2)$$

$$c \in \begin{cases} \pm 1 \pm j & , \quad \pm 1 \pm 3j \\ \pm 3 \pm j & , \quad \pm 3 \pm 3j \end{cases}$$

First, we start off with the first bit, b_0, where we can see from the gray mapping that it will be 0 when the real part of the symbol is negative and it will be 1 when the real part of the symbol is positive. Therefore, we can clearly see that the theoretical LLR for this bit will be defined as shown below:

$$LLR_{b0} = \ln\left(\frac{P(y|\, b_0 = 1)}{P(y|\, b_0 = 0)}\right) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{re}-3}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{re}+3}{\sigma}\right)^2}}\right)$$

We can approximate this LLR by breaking it down into regions. If y_{re} is less than -2, we can clearly see the second term in the numerator will be much less than the first term, so we can ignore it. The same goes for the denominator, but now we can clearly see that the first term will be a lot smaller than the second term. Using these two assumptions, we can approximate the LLR for $y_{re} < -2$ to be the following:

$$LLR_{b0} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}+3}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{re}+3}{\sigma}\right)^2$$

$$LLR_{b0} \approx \frac{4}{\sigma^2}(y_{re}+1) \qquad y_{re} < -2$$

If y_{re} is between -2 and 2, we can clearly see the second term in both the numerator and denominator will be much smaller than the first term, which means we can ignore it. Therefore, the LLR for this region is given with the following expression:

$$LLR_{b0} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2$$

$$LLR_{b0} \approx \frac{2}{\sigma^2}y_{re} \qquad -2 \le y_{re} < 2$$

If y_{re} is greater than 2, we can clearly see the second term in the numerator will be much greater than the first term, so we can ignore the first term. The same goes for the denominator, but now we can clearly see that the first term will be a lot greater than the second term. Using these two assumptions, we can approximate the LLR for $y_{re} > 2$ to be the following:

$$LLR_{b0} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}-3}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{re}-3}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2$$

$$LLR_{b0} \approx \frac{4}{\sigma^2}(y_{re}-1) \qquad y_{re} \ge 2$$

Now, we can move on to the second bit, b_1, where we can see from the gray mapping the bit will be 0 if the real part of the received symbol is greater than 2 or less than -2 and will be 1 otherwise. Therefore, we can clearly see that the theoretical LLR for this bit will be defined as shown below:

$$LLR_{b1} = \ln\left(\frac{P(y|\, b_1 = 1)}{P(y|\, b_1 = 0)}\right) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}-3}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}+3}{\sigma}\right)^2}}\right)$$

We can approximate this LLR by breaking it down into regions. If y_{re} is negative, the contribution from the left terms in the numerator and denominator will be minimal compared to their counterparts, so we can approximate the LLR for this region to be equal to the following:

$$LLR_{b1} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}+3}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{re}+1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{re}+3}{\sigma}\right)^2$$

$$LLR_{b1} \approx \frac{2}{\sigma^2}(y_{re}+2) \qquad y_{re} < 0$$

If y_{re} is positive, the contribution from the right terms in the numerator and denominator will be minimal compared to their counterparts, so we can approximate the LLR for this region to be equal to the following:

$$LLR_{b1} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{re}-3}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{re}-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{re}-3}{\sigma}\right)^2$$

$$LLR_{b1} \approx \frac{2}{\sigma^2}(-y_{re}+2) \qquad y_{re} \geq 0$$

Now, we can move on to the third bit, b_2, where we can see from the gray mapping the bit will be 0 if the imaginary part of the received symbol is positive, and the bit will be 1 if the imaginary part is negative. Therefore, we can clearly see that the theoretical LLR for this bit will be defined as shown below:

$$LLR_{b2} = \ln\left(\frac{P(y|\, b_2=1)}{P(y|\, b_2=0)}\right) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-3}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+3}{\sigma}\right)^2}}\right)$$

We can approximate this LLR by breaking it down into regions. If y_{im} is less than -2, we can clearly see the second term in the numerator will be much less than the first term, so we can ignore it. The same goes for the denominator, but now we can clearly see that the first term will be a lot smaller than the second term. Using these two assumptions, we can approximate the LLR for $y_{im} < -2$ to be the following:

$$LLR_{b2} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+3}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{im}+3}{\sigma}\right)^2$$

$$LLR_{b2} \approx \frac{4}{\sigma^2}(y_{im}+1) \qquad y_{im} < -2$$

If y_{im} is between -2 and 2, we can clearly see the second term in both the numerator and denominator will be much smaller than the first term, which means we can ignore it. Therefore, the LLR for this region is given with the following expression:

$$LLR_{b2} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2$$

$$LLR_{b2} \approx \frac{2}{\sigma^2}y_{im} \qquad -2 \le y_{im} < 2$$

If y_{im} is greater than 2, we can clearly see the second term in the numerator will be much greater than the first term, so we can ignore the first term. The same goes for the denominator, but now we can clearly see that the first term will be a lot greater than the second term. Using these two assumptions, we can approximate the LLR for $y_{im} > 2$ to be the following:

$$LLR_{b2} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-3}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{im}-3}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2$$

$$LLR_{b2} \approx \frac{4}{\sigma^2}(y_{im}-1) \qquad y_{im} \ge 2$$

Finally, we can move on to the last bit, b_3, where we can see from the gray mapping the bit will be 0 if the imaginary part of the received symbol is greater than 2 or less than -2 and will be 1 otherwise. Therefore, we can clearly see that the theoretical LLR for this bit will be defined as shown below:

$$LLR_{b3} = \ln\left(\frac{P(y|\, b_3 = 1)}{P(y|\, b_3 = 0)}\right) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-3}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+3}{\sigma}\right)^2}}\right)$$

We can approximate this LLR by breaking it down into regions. If y_{im} is negative, the contribution from the left terms in the numerator and denominator will be minimal compared to their counterparts, so we can approximate the LLR for this region to be equal to the following:

$$LLR_{b3} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}+3}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{im}+1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{im}+3}{\sigma}\right)^2$$

$$LLR_{b3} \approx \frac{2}{\sigma^2}(y_{im}+2) \qquad y_{im} < 0$$

If y_{im} is positive, the contribution from the right terms in the numerator and denominator will be minimal compared to their counterparts, so we can approximate the LLR for this region to be equal to the following:

$$LLR_{b3} \approx \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{y_{im}-3}{\sigma}\right)^2}}\right) = -\frac{1}{2}\left(\frac{y_{im}-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{im}-3}{\sigma}\right)^2$$

$$LLR_{b3} \approx \frac{2}{\sigma^2}(-y_{im}+2) \qquad y_{im} \geq 0$$

Rayleigh Fading Channel

Rayleigh fading is a specialized model for stochastic fading when there is no line-of-sight propagated signal, where the amplitude gain is characterized by a Rayleigh distribution. The Rayleigh fading channel refers to a multiplicative distortion h(t) of the transmitted signal s(t) and the noise:

$$y(t) = h(t)s(t) + n(t)$$

where y(t) is the received waveform and n(t) is the noise.

Neural Network

The following tables below contain the parameters for the regression-based neural network models simulated for each soft bit of the BPSK, 4QAM, and 16 QAM modulations with varying number of layers for different modulations.

Each of the models use

- a custom Adam optimizer that is useful for both noisy and sparse gradient problems to compute individual adaptive learning rates for different parameters
- ReLU as the activation function for hidden layers (allows models to learn faster and perform better as it prevents the network from behaving as a simple linear unit) and linear as the activation function for the output layer (as values are unbounded)
- a mean squared logarithmic error for the loss function as this regression problem involves calculating the log likelihood ratios
- a mean squared error metric as it is a regression problem
- a truncated normal distribution for the initializer to prevent generation of dead neurons due to the use of ReLU by providing neurons with a slight positive initial bias.
- a fan in fan out topography for the hidden layers as far as feasible

BPSK

The models for this modulation use a mean squared logarithmic error loss function and are evaluated using a mean squared error. A custom Adam optimizer is used with a learning rate = 1, beta_1 = 0.9, decay = 0.99 and does not apply an AMSGrad variant.

10 dB:

The beta_2 parameter for the Adam optimizer has a value of 0.99.

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Random Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.1	1
3	Dense2	(None, 2048)	ReLU	-	-	2
4	Dropout2	(None, 2048)	-	-	0.2	3
5	Dense3	(None, 1024)	ReLU	-	-	4
6	Dropout3	(None, 1024)	-	-	0.4	5
7	Output	(None, 10000)	Linear	-	-	6

15 dB:

The beta_2 parameter for the Adam optimizer has a value of 0.999.

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.1	1
3	Dense2	(None, 2048)	ReLU	Truncated Normal	-	2
4	Dropout2	(None, 2048)	-	-	0.1	3
5	Dense3	(None, 1024)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 1024)	-	-	0.2	5
7	Dense4	(None, 512)	ReLU	Truncated Normal	-	6
8	Dropout4	(None, 512)	-	-	0.4	7
9	Output	(None, 10000)	Linear	-	-	8

20 dB:

The beta_2 parameter for the Adam optimizer has a value of 0.999.

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.1	1
3	Dense2	(None, 2048)	ReLU	Truncated Normal	-	2
4	Dropout2	(None, 2048)	-	-	0.2	3
5	Dense3	(None, 1024)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 1024)	-	-	0.4	5
7	Dense4	(None, 512)	ReLU	Truncated Normal	-	6
8	Dropout4	(None, 512)	-	-	0.6	7
9	Output	(None, 10000)	Linear	-	-	8

4QAM

The models for this modulation use a mean squared logarithmic error loss function and are evaluated using a mean squared error. A custom Adam optimizer is used with a learning rate = 1, beta_1 = 0.9, beta_2 =0.999, decay = 0.99 and does not apply an AMSGrad variant.

10 dB:

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.1	1
3	Dense2	(None, 1024)	ReLU	Truncated Normal	-	1
4	Dropout2	(None, 1024)	-	-	0.2	3
5	Dense3	(None, 256)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 256)	-	-	0.4	5
7	Dense4	(None, 512)	ReLU	Truncated Normal	-	4
8	Dropout4	(None, 512)	-	-	0.8	7
9	Output0	(None, 10000)	Linear	Truncated Normal	-	8
10	Output1	(None, 10000)	Linear	Truncated Normal	-	8

15 dB:

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.2	1
3	Dense2	(None, 2048)	ReLU	Truncated Normal	-	1
4	Dropout2	(None, 2048)	-	-	0.2	3
5	Dense3	(None, 512)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 512)	-	-	0.4	5
7	Dense4	(None, 256)	ReLU	Truncated Normal	-	6
8	Dropout4	(None, 256)	-	-	0.4	7
9	Output0	(None, 10000)	Linear	Truncated Normal	-	8
10	Output1	(None, 10000)	Linear	Truncated Normal	-	8

20 dB:

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.1	1
3	Dense2	(None, 2048)	ReLU	Truncated Normal	-	2
4	Dropout2	(None, 2048)	-	-	0.2	3
5	Dense3	(None, 1024)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 1024)	-	-	0.2	5
7	Dense4	(None, 512)	ReLU	Truncated Normal	-	6
8	Dropout4	(None, 512)	-	-	0.4	7
9	Dense5	(None, 256)	ReLU	Truncated Normal	-	8
10	Dropout5	(None, 256)	-	-	0.6	9
11	Output0	(None, 10000)	Linear	Truncated Normal	-	10
12	Output1	(None, 10000)	Linear	Truncated Normal	-	10

16QAM

The models for this modulation use a mean squared logarithmic error loss function and are evaluated using a mean squared error. A custom Adam optimizer is used with beta_1 = 0.9, beta_2 =0.999, decay = 0.99 and does not apply an AMSGrad variant.

10 dB: The learning rate parameter for the Adam optimizer has a value of 2.

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.1	1
3	Dense2	(None, 2048)	ReLU	Truncated Normal	-	2
4	Dropout2	(None, 2048)	-	-	0.2	3
5	Dense3	(None, 1024)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 1024)	-	-	0.3	5
7	Dense4	(None, 512)	ReLU	Truncated Normal	-	6
8	Dropout4	(None, 512)	-	-	0.3	7
9	Dense5	(None, 256)	ReLU	Truncated Normal	-	8
10	Dropout5	(None, 256)	-	-	0.4	9
11	Dense6	(None, 128)	ReLU	Truncated Normal	-	10
12	Dropout6	(None, 128)	-	-	0.5	11
13	Dense7	(None, 512)	ReLU	Truncated Normal	-	12
14	Dropout7	(None, 512)	-	-	0.7	13
15	Output0	(None, 10000)	Linear	Truncated Normal	-	13
16	Output1	(None, 10000)	Linear	Truncated Normal	-	13

15 dB: The learning rate parameter for the Adam optimizer has a value of 1.

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.2	1
3	Dense2	(None, 2048)	ReLU	Truncated Normal	-	2
4	Dropout2	(None, 2048)	-	-	0.2	3
5	Dense3	(None, 1024)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 1024)	-	-	0.2	5
7	Dense4	(None, 512)	ReLU	Truncated Normal	-	6
8	Dropout4	(None, 512)	-	-	0.2	7
9	Dense5	(None, 256)	ReLU	Truncated Normal	-	8
10	Dropout5	(None, 256)	-	-	0.2	7
11	Dense6	(None, 128)	ReLU	Truncated Normal	-	10
12	Dropout6	(None, 128)	-	-	0.3	11
13	Output0	(None, 10000)	Linear	Truncated Normal	-	12
14	Output1	(None, 10000)	Linear	Truncated Normal	-	12

20 dB: The learning rate parameter for the Adam optimizer has a value of 1.

#	Layer	Shape	Activation Function	Initializer	Rate	Link
0	Input	(None, 20000)	-	-	-	
1	Dense1	(None, 4096)	ReLU	Truncated Normal	-	0
2	Dropout1	(None, 4096)	-	-	0.2	1
3	Dense2	(None, 2048)	ReLU	Truncated Normal	-	2
4	Dropout2	(None, 2048)	-	-	0.2	3
5	Dense3	(None, 1024)	ReLU	Truncated Normal	-	4
6	Dropout3	(None, 1024)	-	-	0.2	5
7	Dense4	(None, 512)	ReLU	Truncated Normal	-	6
8	Dropout4	(None, 512)	-	-	0.3	7
9	Dense5	(None, 256)	ReLU	Truncated Normal	-	8
10	Dropout5	(None, 256)	-	-	0.6	9
11	Output0	(None, 10000)	Linear	Truncated Normal	-	10
12	Output1	(None, 10000)	Linear	Truncated Normal	-	10

III. PROCEDURES

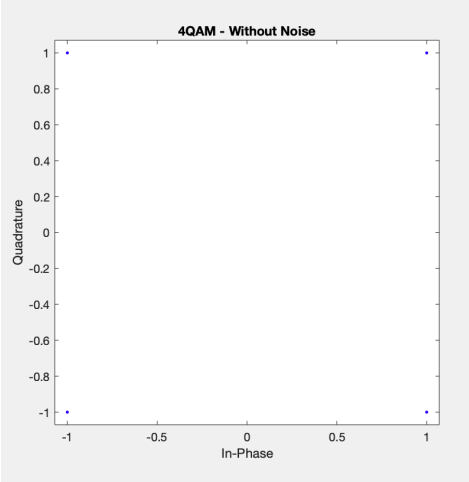
Communications

In order to train the neural network, we must first generate and simulate the transmission of symbols through an AWGN channel, which was done using a wide range of SNRs from $0\ to\ 25\ dB$ in increments of $5\ dB$.

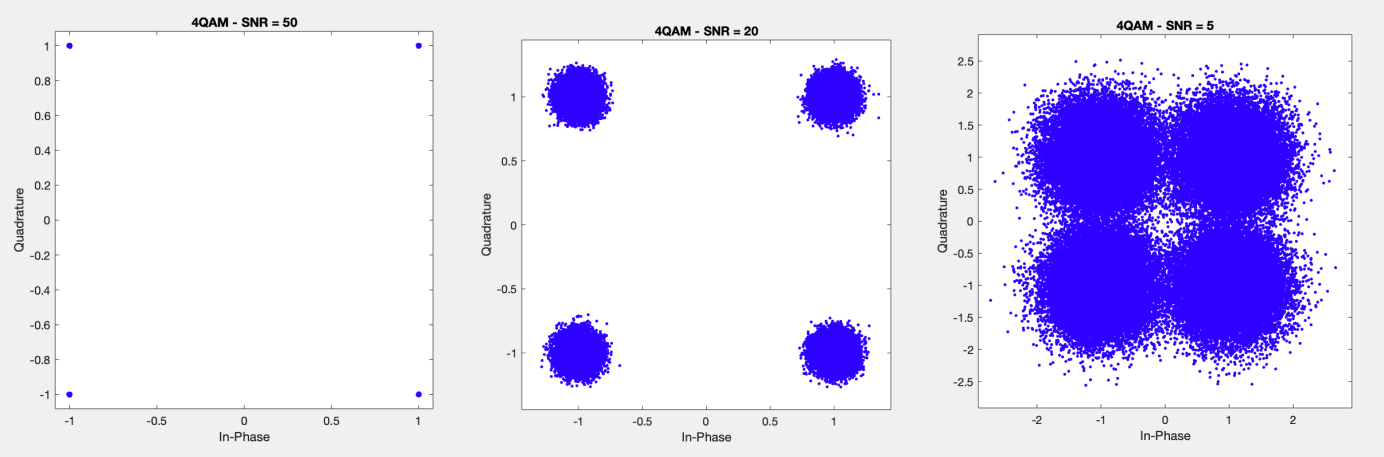
- First, we must generate several symbols (e.g. 100,000), which was done using the `randi` function in MATLAB. The random symbols were generated from 0 to $M-1$, where M is dependent on the type of modulation (e.g. $4QAM$ has $M = 4$). The number of bits transmitted per symbol is given by the following expression: $k = \log_2 M$
- The randomly generated symbols are to be converted to bits using the `de2bi()` function, which we would use to compare with the received bits, in order to assess the Bit Error Rate of the channel.
- The symbols were then modulated using the function `qammod()` for QAM and `pskmod()` for PSK. For QAM, the function automatically uses gray coding when modulating the symbols, which will be shown in the following section where the theoretical LLRs are computed for each modulation. For PSK, the default function does not use gray coding, but the `'gray'` argument was passed inside `pskmod()`, which made sure to use gray coding when modulating the symbols.
- The next step would be to pass the modulated symbols through an AWGN channel using the function `agwn()` that is dependent on the signal to noise ratio of the channel.
- After passing the modulated signals through the channel, which effectively added white gaussian noise to each symbol, the LLRs for each soft bit were computed in MATLAB.
- The received noisy modulated symbols are then demodulated using `qamdemod()` and `pskdemod()`.
- The demodulated signals are now converted into bits using the `de2bi()` function in order to compare with the original transmitted bits in order to assess the quality of the channel.
- The number of errors for each SNR was then calculated using the `biterr()` function that took the transmitted bits and the received bits as inputs. The number of errors was then divided by the total number of bits transmitted in order to find the Bit Error Rate (BER) for each SNR.

4QAM Simulation Details and SNR vs. BER

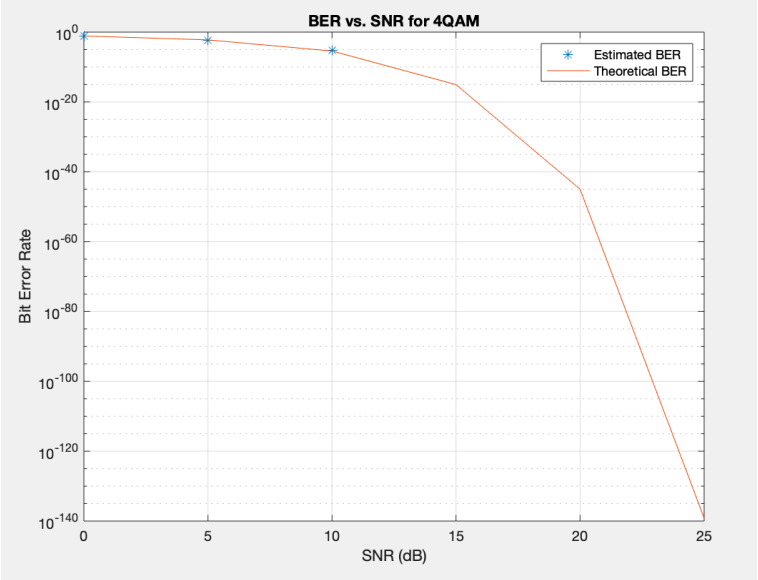
The 4QAM constellation without noise is shown below:



The received symbols for three different SNRs are shown below:



It is clear that for higher values of SNR, the received constellation behaved more and more like the theoretical constellation. Therefore, we expect a very low bit error rate as the SNR increases. The simulated results compared to the theoretical expectations are depicted in the plot and table shown below:

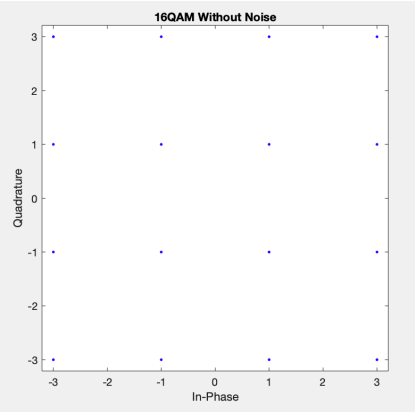


SNR	Simulated BER	Theoretical BER
0	0.07934	0.07865
5	0.00617	0.00595
10	0	3.8721×10^{-6}
15	0	9.1240×10^{-16}
20	0	1.0442×10^{-45}
25	0	7.3070×10^{-140}

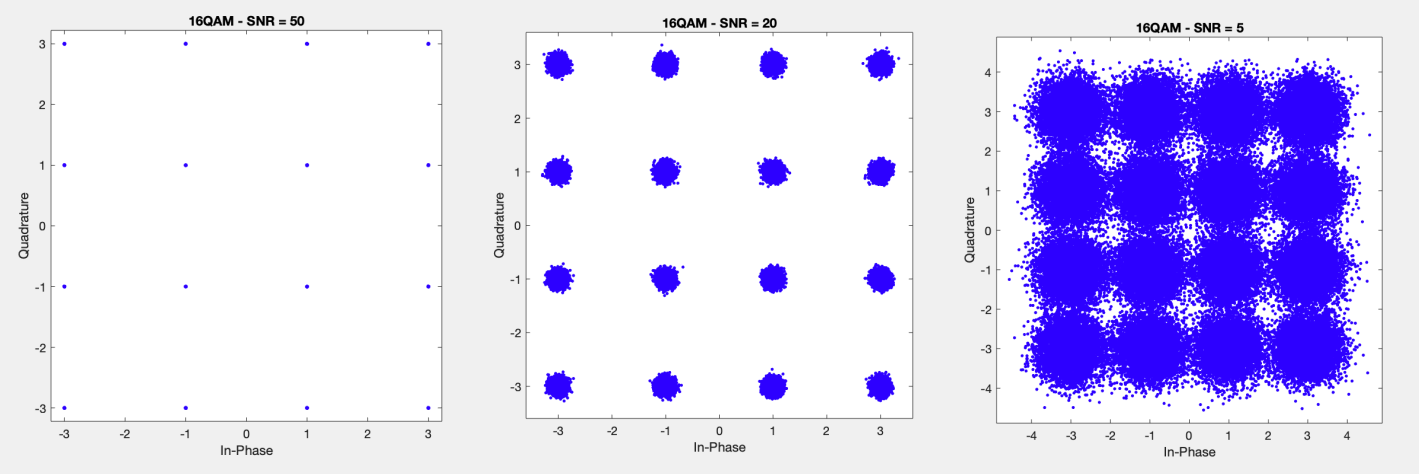
Both the plot and the table show that the simulated results are well within expectation. The plot does not have data points for when the simulated BER becomes 0, so it is much easier to compare the two in the table. It is not surprising that the simulated BER is zero for SNR values of 10 and larger as we only ran the simulation for one hundred thousand symbols. If we had transmitted millions and millions of bits, then we would expect the simulated BER to be a number in the range of the theoretical BER. However, running the simulation for millions of bits is not possible as MATLAB cannot support such a large action and it would take hours to run, if not completely crashing.

16QAM Simulation Details and SNR vs. BER

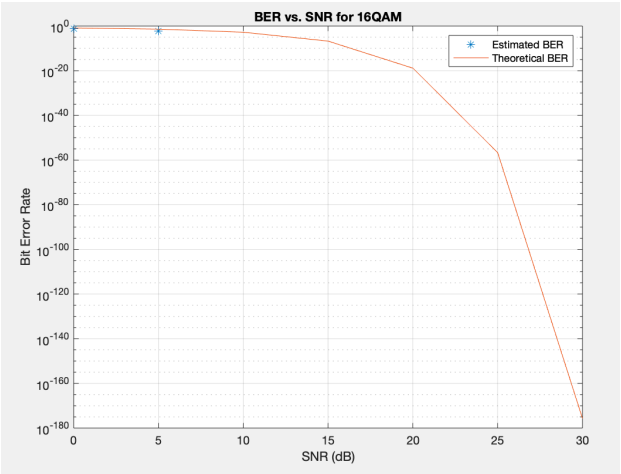
The 16QAM constellation without noise is shown below:



The received symbols for three different SNRs are shown below:



It is clear that for higher values of SNR, the received constellation behaved more and more like the theoretical constellation. Therefore, we expect a very low bit error rate as the SNR increases. The simulated results compared to the theoretical expectations are depicted in the plot and table shown below:

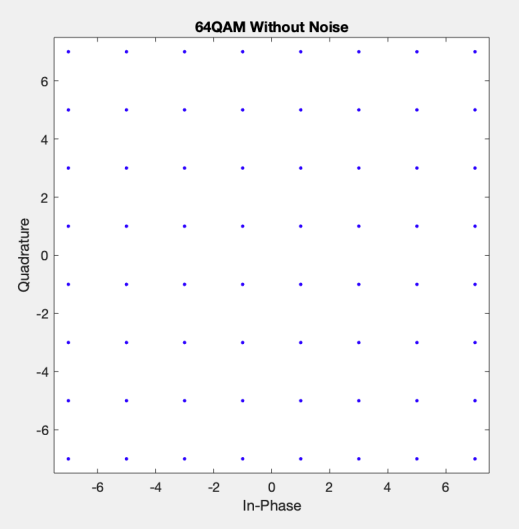


SNR	Simulated BER	Theoretical BER
0	0.05914	0.14098
5	0.00449	0.04189
10	0	0.00175
15	0	1.8419×10^{-7}
20	0	1.4040×10^{-19}
25	0	2.1794×10^{-57}

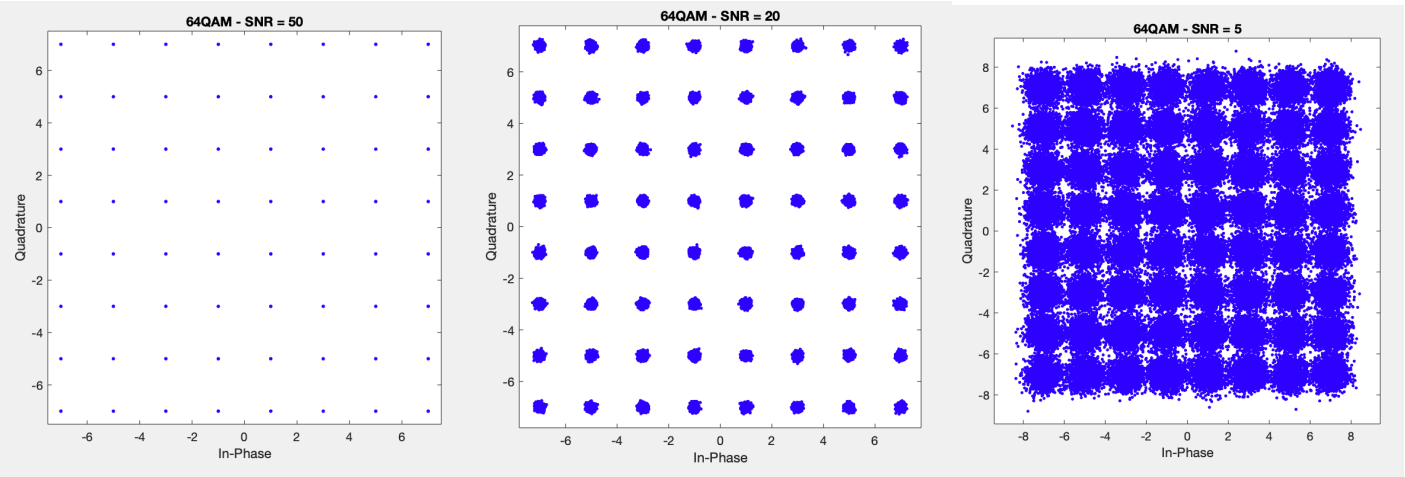
The results actually show that our simulated model had a lower simulated BER than what was expected, which although it is surprising, it is quite a good result.

64QAM Simulation Details and SNR vs. BER

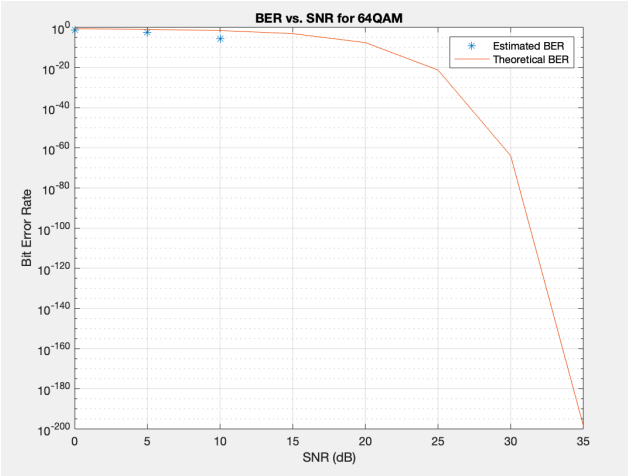
The 64QAM constellation without noise is shown below:



The received symbols for three different SNRs are shown below:



It is clear that for higher values of SNR, the received constellation behaved more and more like the theoretical constellation. Therefore, we expect a very low bit error rate as the SNR increases.

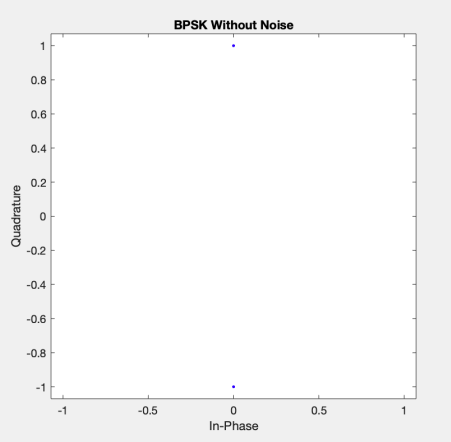


SNR	Simulated BER	Theoretical BER
0	0.04553	0.19984
5	0.00342	0.10079
10	1.6667×10^{-6}	0.02653
15	0	0.00077
20	0	2.6339×10^{-8}
25	0	5.8178×10^{-22}

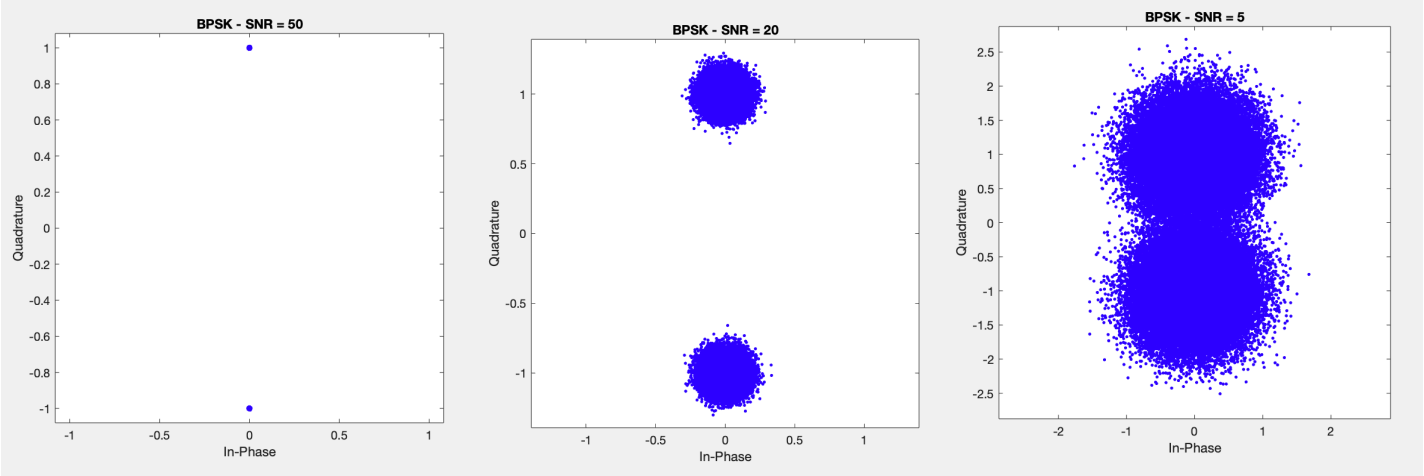
The results actually show that our simulated model had a much lower simulated BER than what was expected, which although it is surprising, it is quite a good result.

BPSK Simulation Details and SNR vs. BER

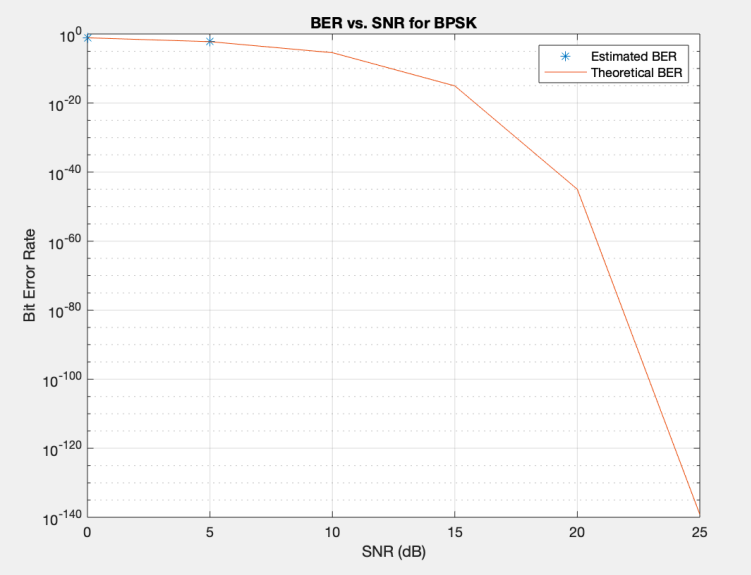
The BPSK constellation without noise is shown below:



The received symbols for three different SNRs are shown below:



It is clear that for higher values of SNR, the received constellation behaved more and more like the theoretical constellation. Therefore, we expect a very low bit error rate as the SNR increases. The simulated results compared to the theoretical expectations are depicted in the plot and table shown below:

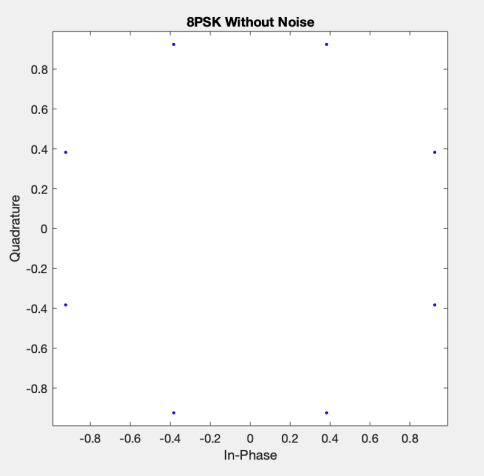


SNR	Simulated BER	Theoretical BER
0	0.07942	0.07865
5	0.00623	0.00595
10	0	3.8721×10^{-6}
15	0	9.1240×10^{-16}
20	0	1.0442×10^{-45}
25	0	7.3070×10^{-140}

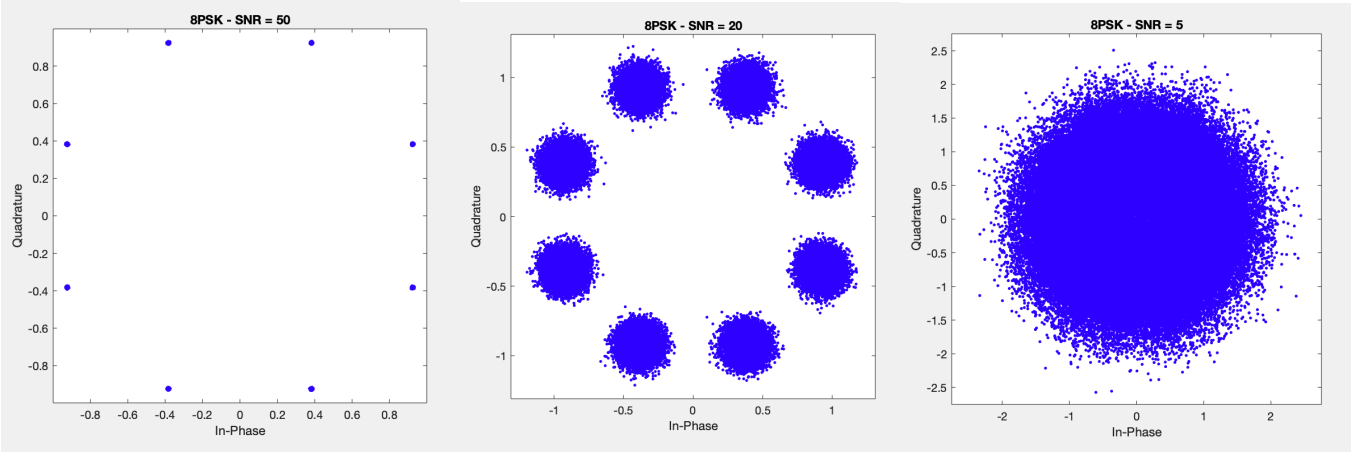
The figure clearly shows that the bit error rate behaves as expected. Our experimental bit error rate becomes zero at 10 dB, which is expected as the theoretical BER is very small for large SNR values.

8PSK Simulation Details and SNR vs. BER

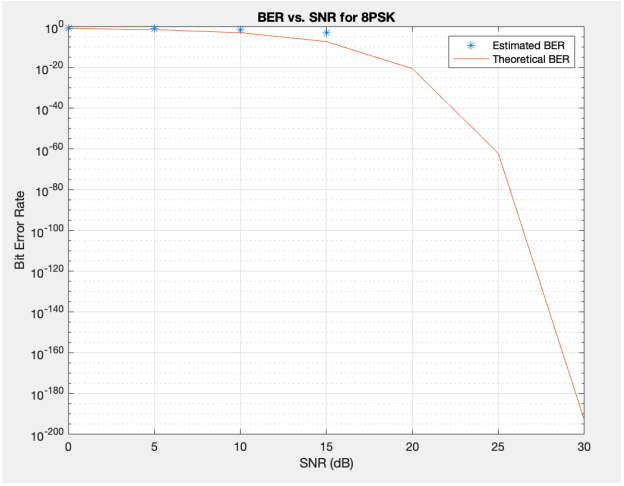
The 8PSK constellation without noise is shown below:



The received symbols for three different SNRs are shown below:



It is clear that for higher values of SNR, the received constellation behaved more and more like the theoretical constellation. Therefore, we expect a very low bit error rate as the SNR increases. The simulated results compared to the theoretical expectations are depicted in the plot and table shown below:



SNR	Simulated BER	Theoretical BER
0	0.24052	0.12269
5	0.11799	0.03186
10	0.02921	0.00101
15	0.00072	4.5161×10^{-8}
20	0	2.3324×10^{-21}
25	0	7.3146×10^{-63}

The results show that our model actually resulted in a much higher bit error rate than expected. However, the channel still behaves normally as the BER dropped significantly as SNR increased.

Hamming Code

In order to better improve the bit error rate of the channel, we decided to incorporate Hamming Block Coding, which encodes a message, making it more resistant to errors. The Hamming method takes a message of length k and encodes it to a codeword length of n. The relationship between the message length and codeword length are shown below:

$$n = 2^m - 1$$

$$k = 2^m - m - 1$$

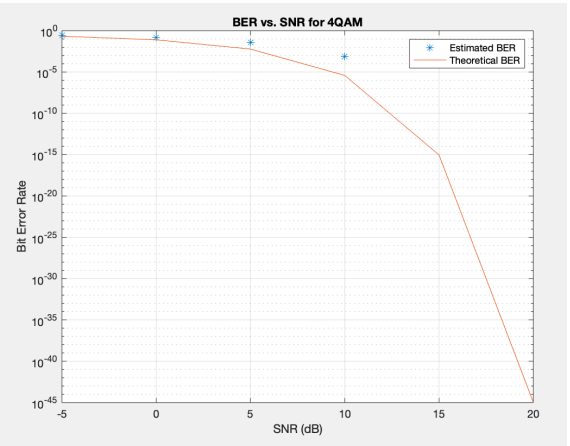
$$m \geq 2$$

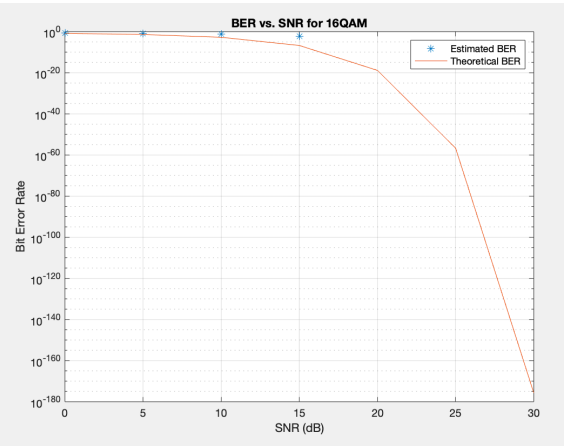
The parameter m essentially determines the amount of bits/symbols we want to transmit over the channel. This parameter was chosen differently for each simulation due to MATLAB taking a long time to carry out large bit transmissions. The modulations chosen for this task were the modulations which had LLRs generated in order for neural network analysis.

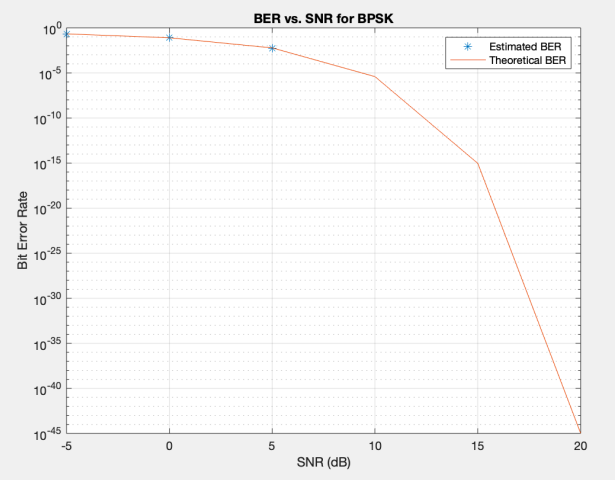
The chart below shows the different values of m used for each modulation:

Modulation	m
4QAM	13
16QAM	12
BPSK	

The BER vs. SNR plots for each modulation are shown below:







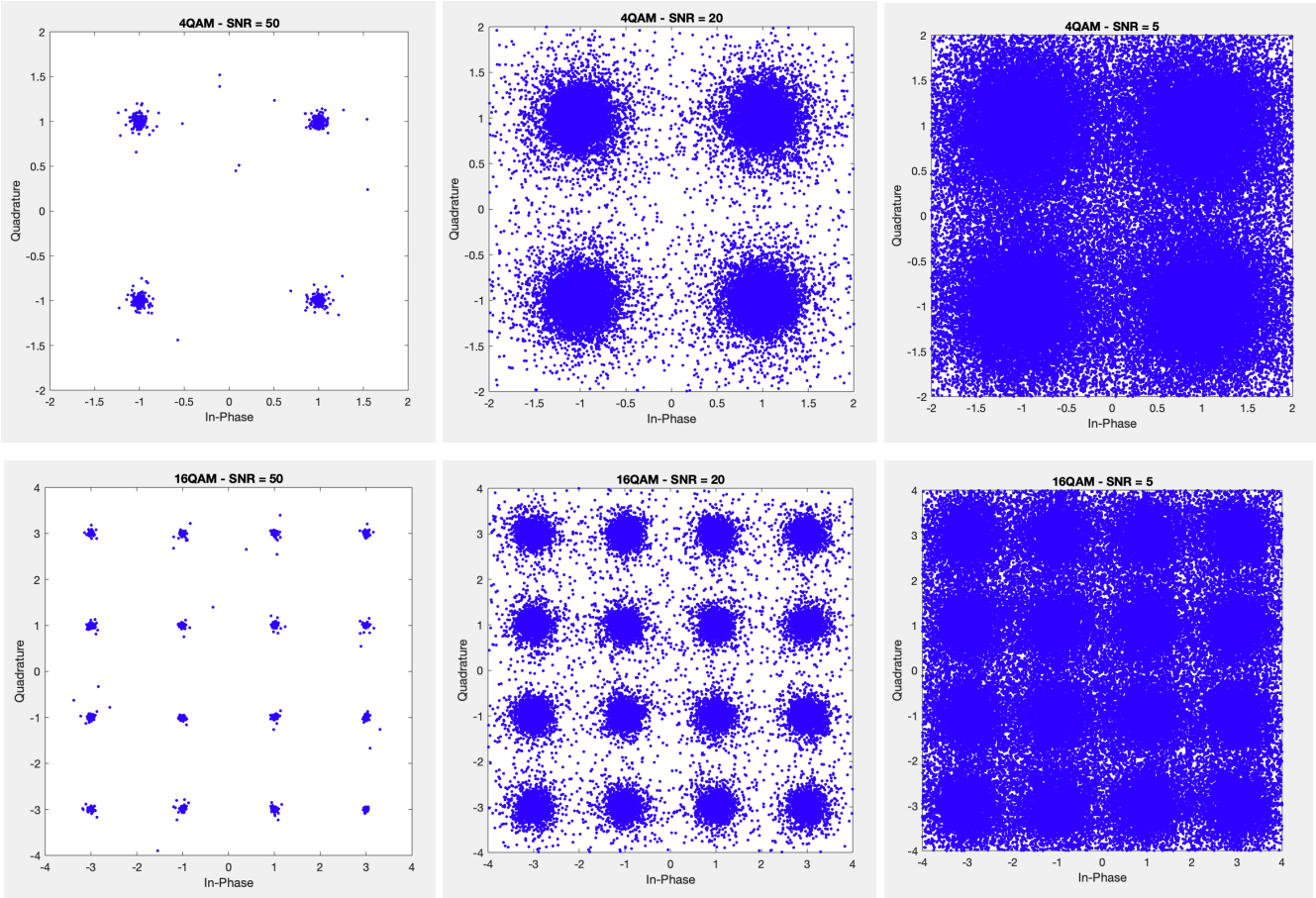
The Hamming Code did not really help the SNR vs. BER whatsoever, in fact it made them worse. This is probably due to the fact that we could not transmit many symbols over the channel as we could in the previous simulation, which was due to MATLAB constraints. Theoretically, channel encoding should help ensure that fewer errors are made in the transmission, but in order to properly test this out, we would need to transmit many more bits to actually get a good estimation of the bit error rate.

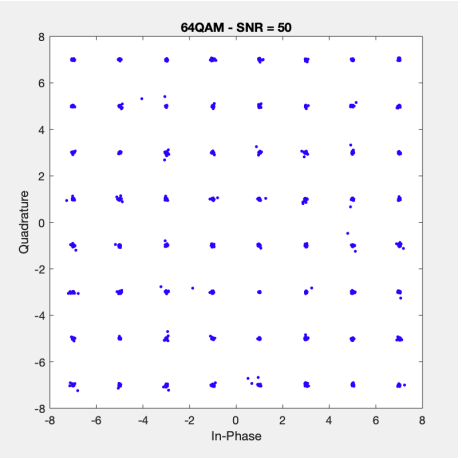
Rayleigh Fading Channel

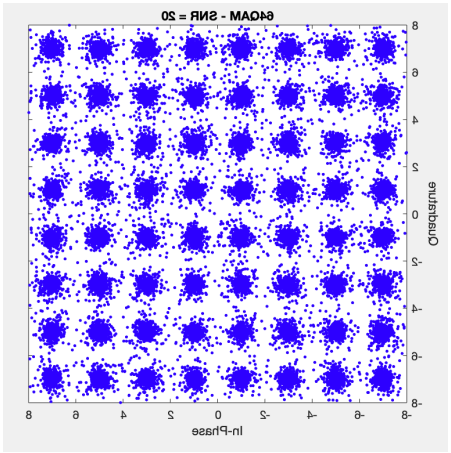
The Rayleigh Fading Channel is a model used for the effect of a propagation environment on a radio signal, mostly used for wireless communication. It is seen as a reasonable model for showing the effect that heavily built-up urban environments have on radio signals. Therefore, it would be interesting to simulate the channel in this project as a Rayleigh Fading Channel and not just AWGN.

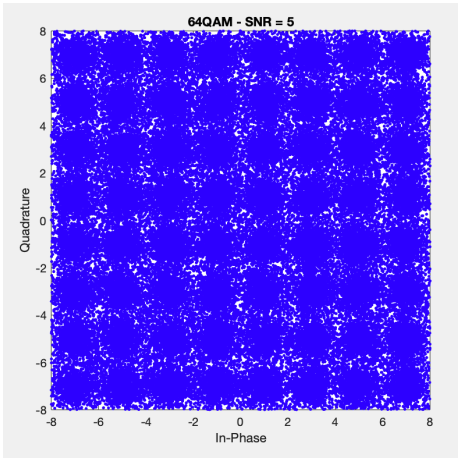
For this part of the project, we will assume that we do not want or do not know how to compute the LLRs. We assumed that in addition to the AWGN, the symbols are multiplied by a Rayleigh-distributed amplitude and phase shifted by a random phase uniformly distributed over $(0,2\pi)$. To simulate the effects of a Rayleigh flat-fading channel, the symbols are amplified by the channel (h) and AWGN noise is added before modulation where $h = \frac{1}{\sqrt{2}}(x + jy)$ (x and y are normally distributed random numbers). To equalize these received symbols, they are then divided by h. The theoretical modulation is calculated using `berfading()`.

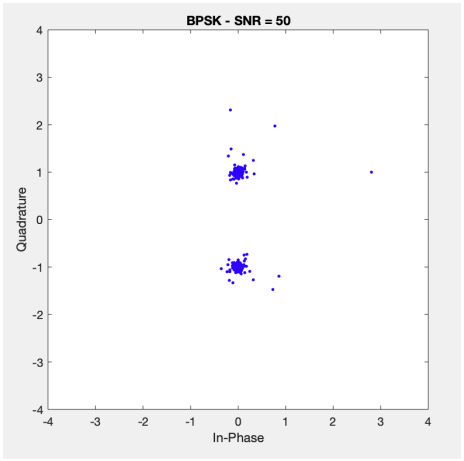
The received symbols for each type of modulation as well as multiple SNRs are shown below:

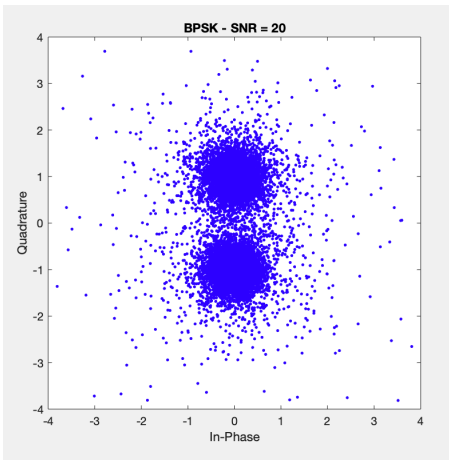


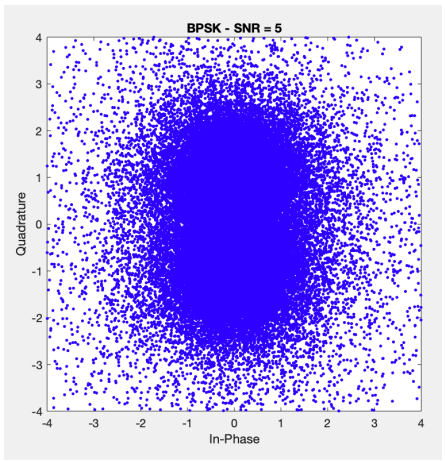


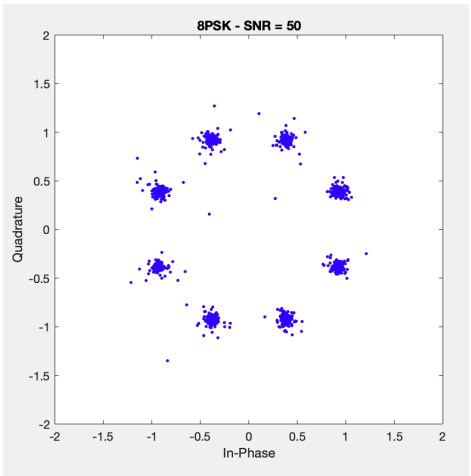


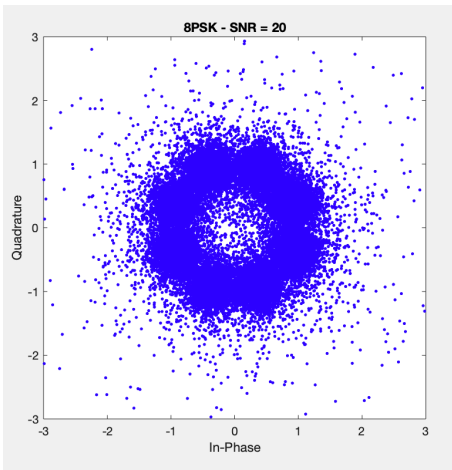


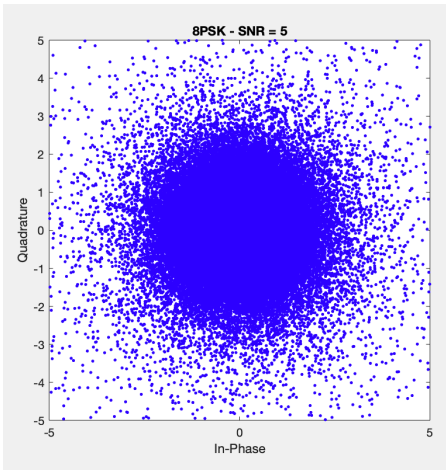




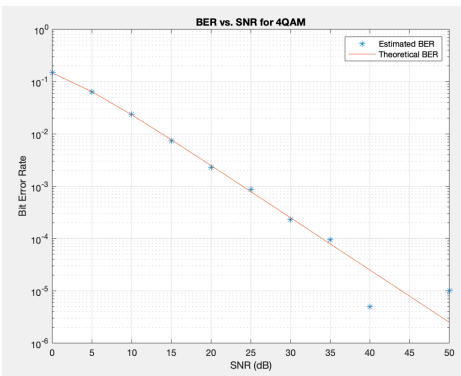


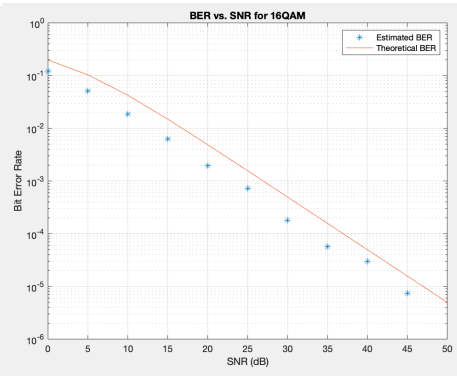


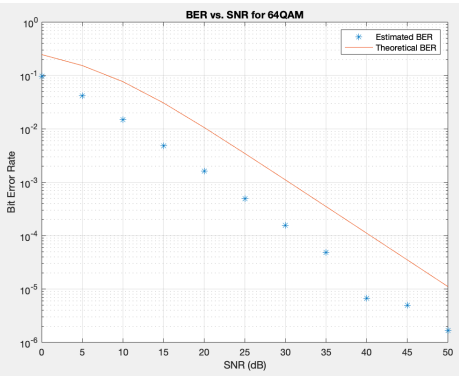


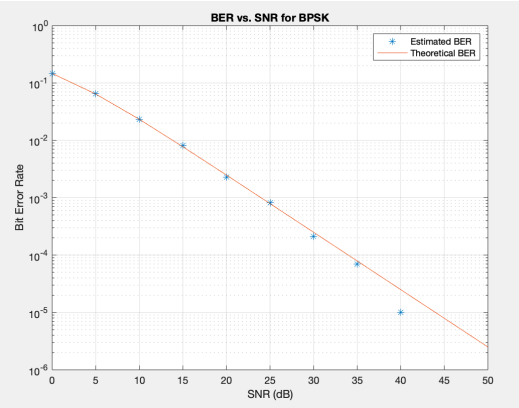


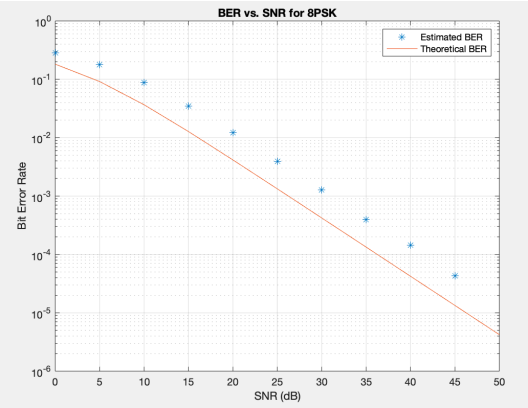
The simulated and theoretical BER vs SNR curves for each modulation are shown below:







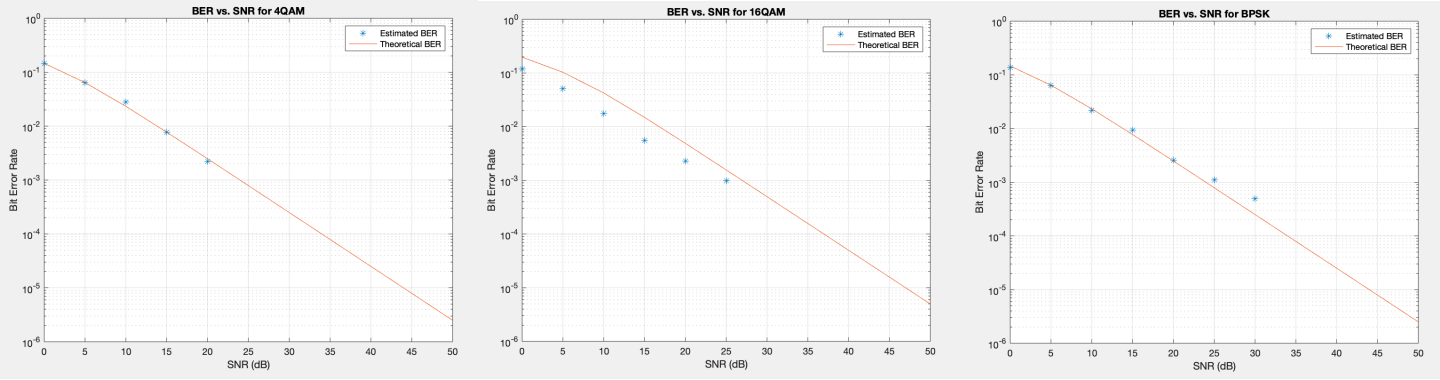




At first glance, it is easy to see that the Rayleigh Facing Channel naturally produces a larger BER for every SNR when compared to the AWGN channel. For the higher order quadrature amplitude modulations through the AWGN channel, the simulated BER was lower than the theoretical SNR, which is also seen in the Rayleigh Fading Channel. For the higher order phase shift keying modulation through the AWGN channel, the simulated BER was higher than the theoretical SNR, which is also seen in the Rayleigh Fading Channel.

Rayleigh Fading Channel with Hamming Block Coding

In order to better see the effect of Hamming Code on the bit transmission, we decided to also perform it on the Rayleigh Fading Channel and compare the results to the AWGN channel. This was done for 4QAM, 16QAM, as well as BPSK, and the same procedures as before were followed. The resulting SNR vs. BER plots are shown below:



The results show once again that adding the Hamming Code does not improve the BER, which is contrary to what we expected. The BER for very large SNR's does go to zero for the RFC with Hamming Code than just the pure RFC because there were fewer bits transmitted due to MATLAB's constraints.

Machine Learning

To estimate LLRs for different modulations and SNR values, we create nine different neural net models for BPSK, 4QAM and 16QAM for the SNR values of 10 dB, 15dB and 20dB.

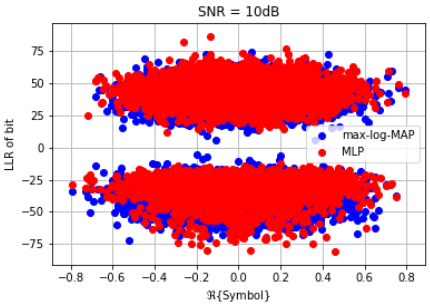
- The received symbols generated from MATLAB are first divided into their real and imaginary components, sym_r and sym_i, before they are fed as multiple inputs into our neural network.
- The MATLAB data containing both the LLRs and the received symbols generated for each digital modulation are shuffled and split into four equal datasets for training, validation, to evaluate and test the score of the model, and finally for a given symbol to estimate its LLRs per bit and compare these values to the theoretically calculated values.
- After preprocessing of the data is completed, the training dataset (indexed at 0) and the validation dataset (indexed at 1) are passed into the neural network model, defined in `llr_model()`, for training purposes and to provide insight into whether our neural network model is overfitting or not using the `keras.model.fit` function. Additionally, this is used to plot the training loss+error to evaluate the efficiency of our model.
- Our neural network model was then evaluated with our scoring dataset (indexed at 2) using the keras.model.evaluate function, where the scalar test loss and the mean squared error was returned to inspect the performance of our neural network model and test its generalizability.
- In order to generate output predictions of our LLRs for our input symbols (indexed at 3), we used the keras.model.predict function. The estimated LLRs (i.e. pred_llrs[0]) were plotted against the theoretically calculated LLRs (i.e. LLR[3]) to visualize the performance and accuracy of our neural network model.
- Additionally, the plots of the LLR, derived via max-log-MAP and the neural network model, as a function of the real part of the received symbols were plotted as another visual indicator of our neural network's performance.
- The whole process is then repeated for each digital modulation tested, with the only difference being the MATLAB data generated for each modulation, and the neural network architecture defined in llr_model(). Specifically, BPSK uses a multi-input single-output (MISO) neural network, while 4QAM and 16QAM utilizes a multi-input multi-output (MISO) neural network.

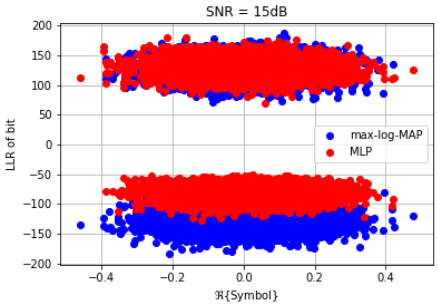
IV. PREDICTION RESULTS

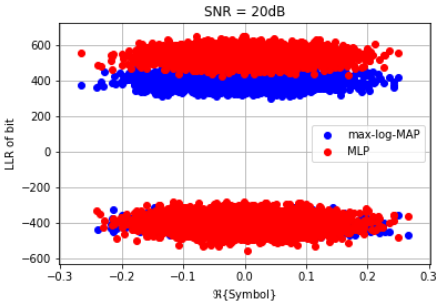
A. Single SNR for a Given Modulation

The following figures demonstrate the relationship between the LLRs predicted by our Multilayer Perceptron (MLP) network and the max-log-MAP calculated based on their SNR values (10 dB, 15 dB and 20 dB). As expected, for higher SNRs, the predicted values closely estimate their expected values.

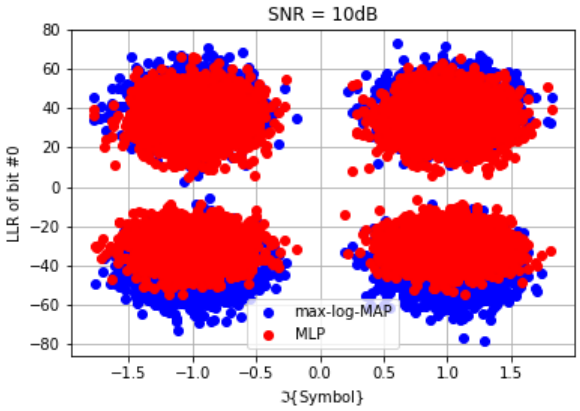
BPSK

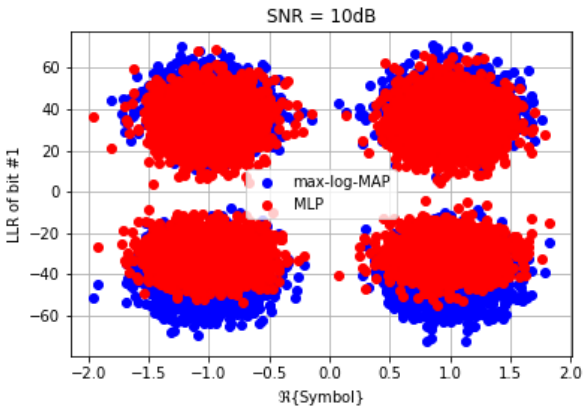


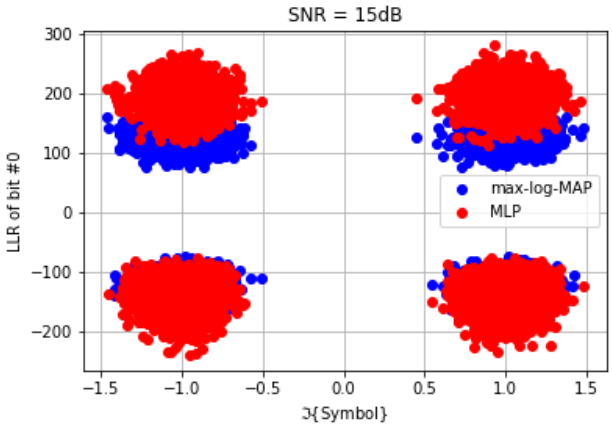


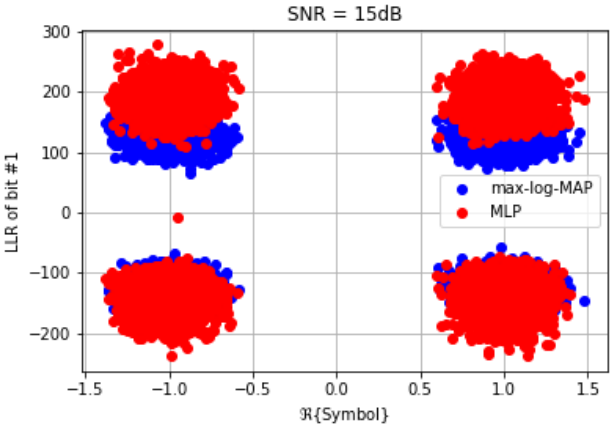


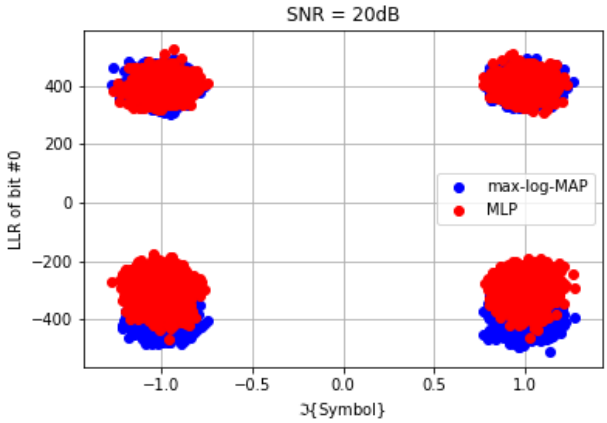
4QAM

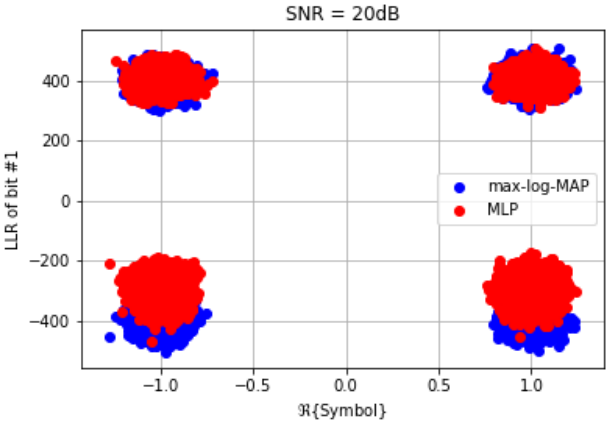




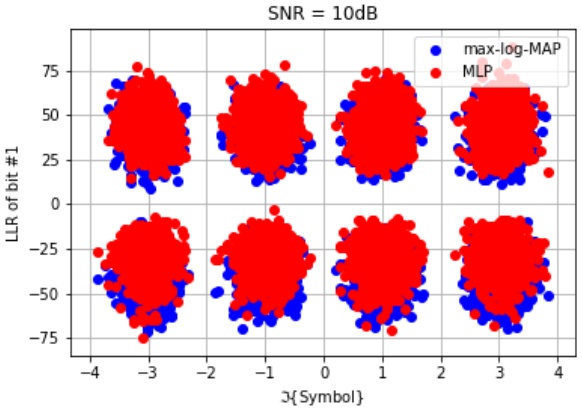


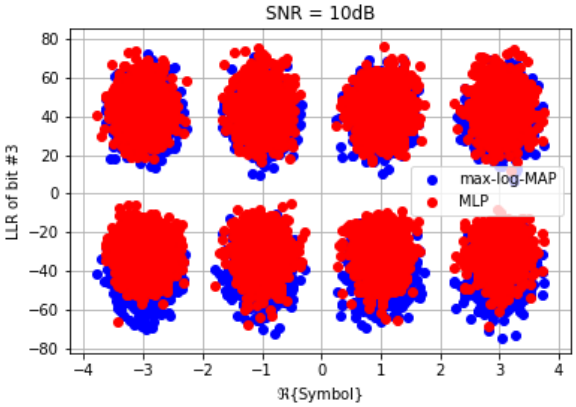


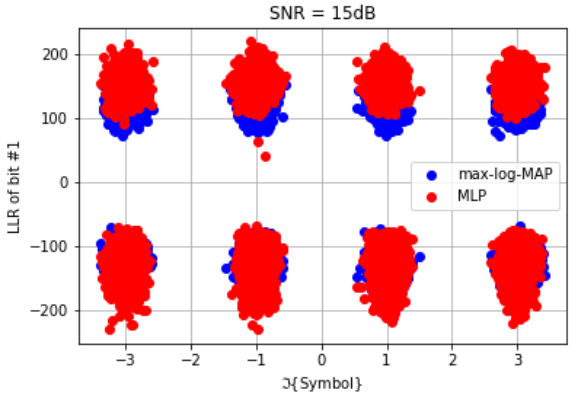

SNR = 20dB
LLR of bit #0
ℑ{Symbol}
max-log-MAP
MLP

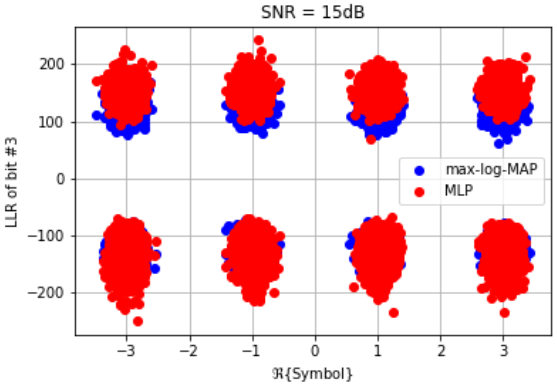

SNR = 20dB
LLR of bit #1
ℜ{Symbol}
max-log-MAP
MLP

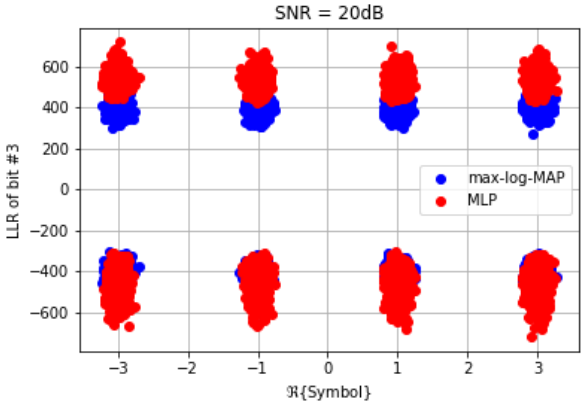
16QAM

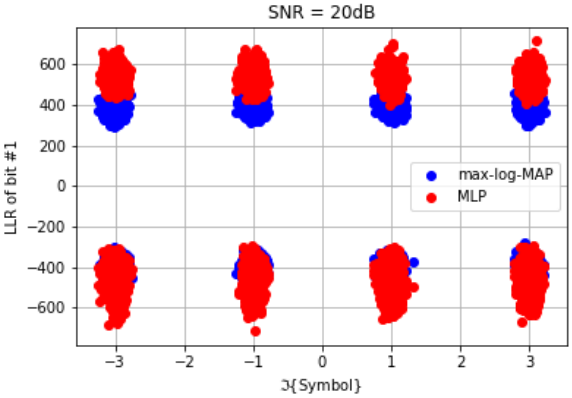

SNR = 10dB
LLR of bit #1
ℑ{Symbol}
max-log-MAP
MLP


SNR = 10dB
LLR of bit #3
ℜ{Symbol}
max-log-MAP
MLP


SNR = 15dB
LLR of bit #1
ℑ{Symbol}
max-log-MAP
MLP

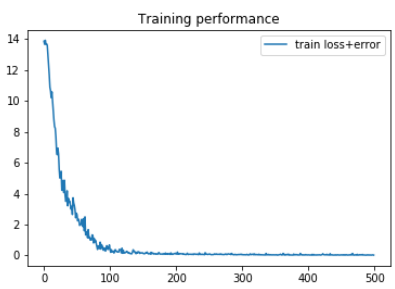

SNR = 15dB
LLR of bit #3
ℜ{Symbol}
max-log-MAP
MLP


SNR = 20dB
LLR of bit #3
ℜ{Symbol}
max-log-MAP
MLP

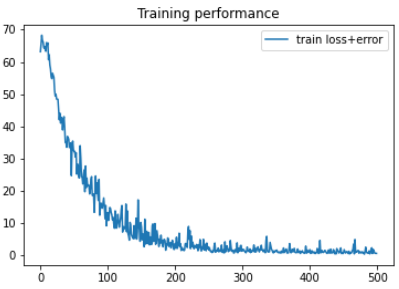

SNR = 20dB
LLR of bit #1
ℑ{Symbol}
max-log-MAP
MLP

Performance

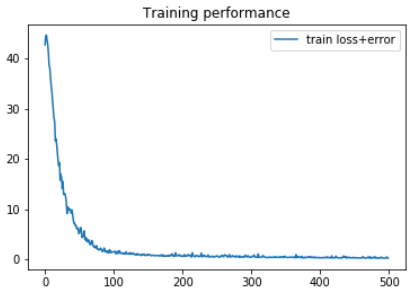
The following figures demonstrate the performance of the models for a given SNR value (20 dB) for all three modulations against the number of epochs for training (500). Thus, neural networks are a much more efficient way of estimating the LLR of a bit of a symbol giving the modulation and SNR value.



BPSK



4QAM

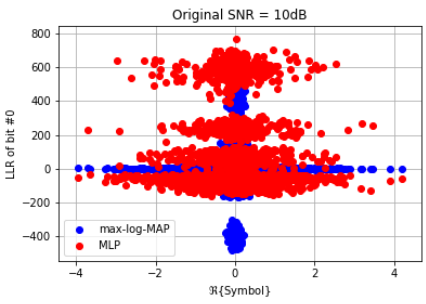


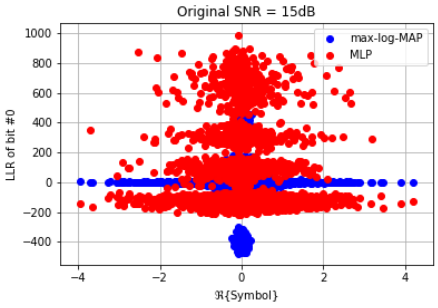
16QAM

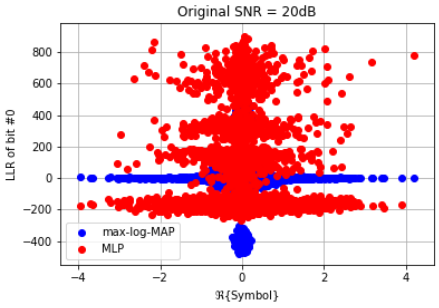
B. Mixed SNRs for a Given Modulation

For this section, a set of symbols are generated from five different SNR values (0 dB, 5 dB, 10 dB, 15 dB and 20 dB) and their respective max-log-MAP values are calculated. The following figures demonstrate the relationship between the LLRs for these symbols predicted by our Multilayer Perceptron (MLP) network for a single SNR value with the actual max-log-MAP as mentioned above. As expected, for higher SNRs, the predicted values better estimate their expected values. However, due to the nature of the neural network, it is not a very useful estimate and centers around an LLR value of 0. The models for BPSK show the greatest accuracy at predicting LLRS with different SNRs.

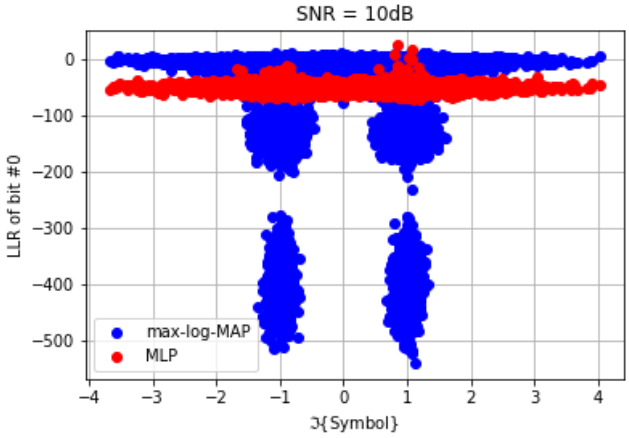
BPSK

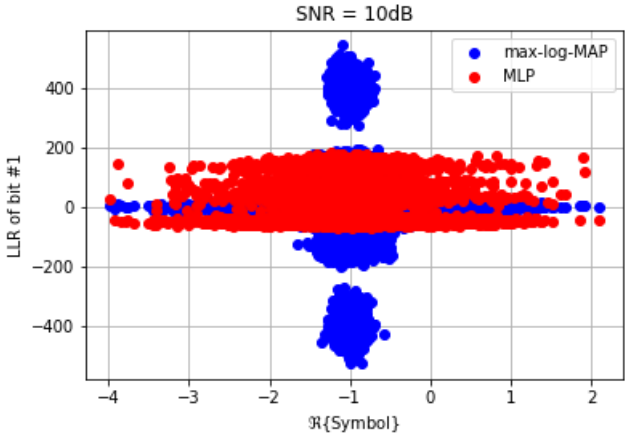


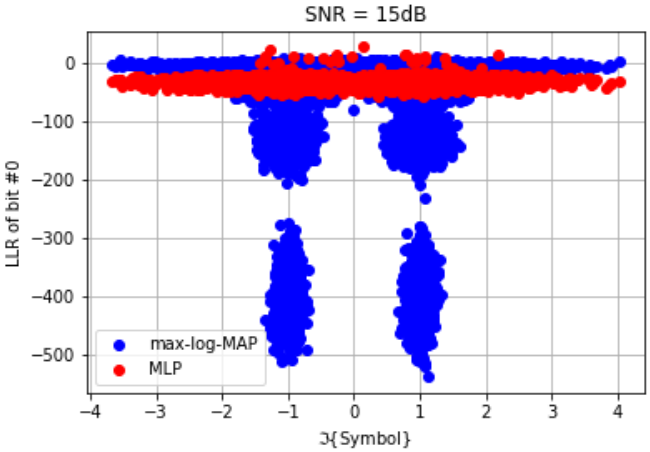


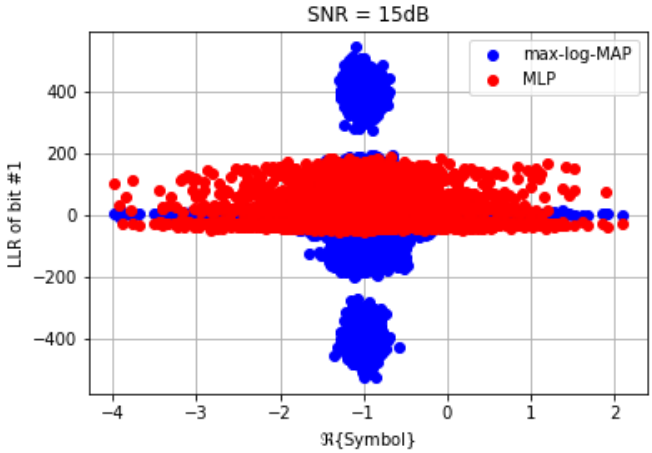


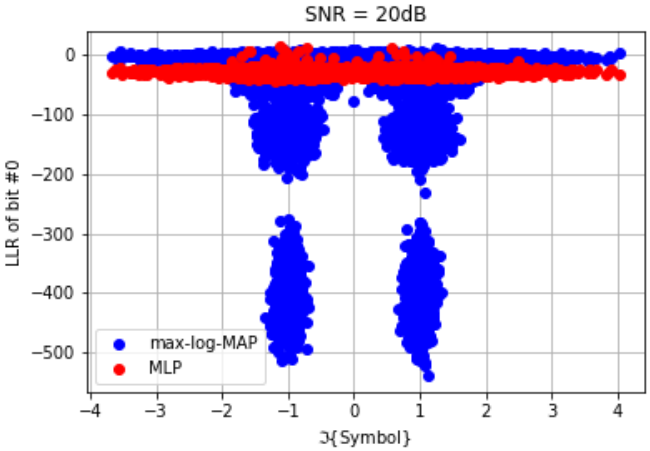
4QAM

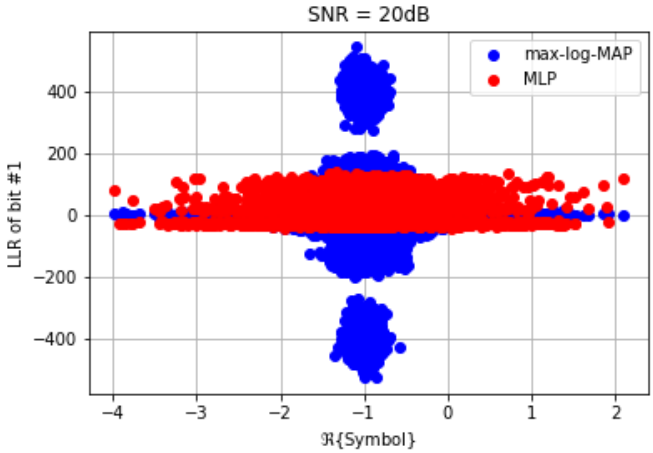

SNR = 10dB
LLR of bit #0
ℑ{Symbol}
max-log-MAP
MLP


SNR = 10dB
LLR of bit #1
ℜ{Symbol}
max-log-MAP
MLP

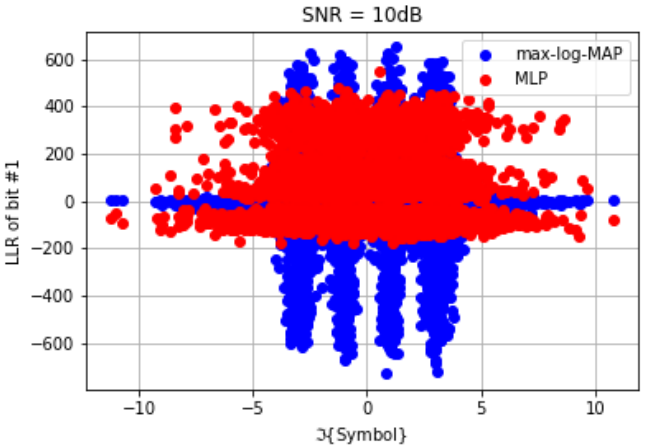

SNR = 15dB
LLR of bit #0
ℑ{Symbol}
max-log-MAP
MLP

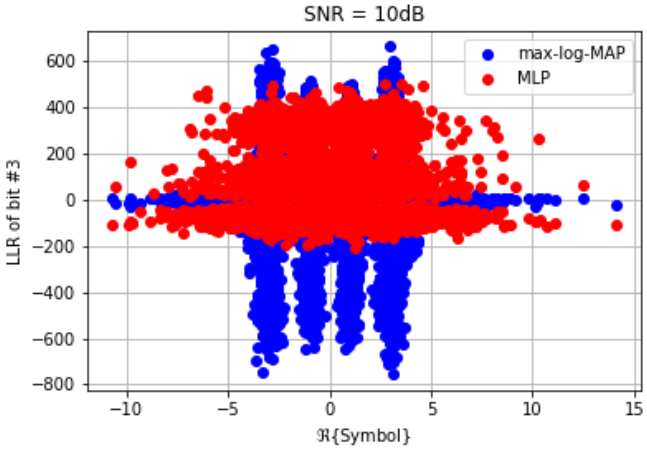

SNR = 15dB
LLR of bit #1
ℜ{Symbol}
max-log-MAP
MLP

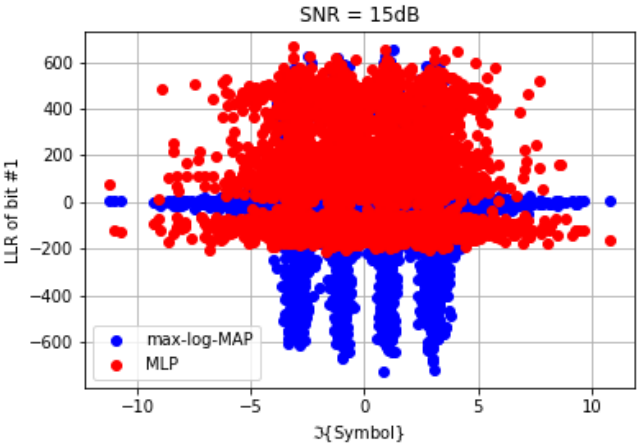

SNR = 20dB
LLR of bit #0
ℑ{Symbol}
max-log-MAP
MLP

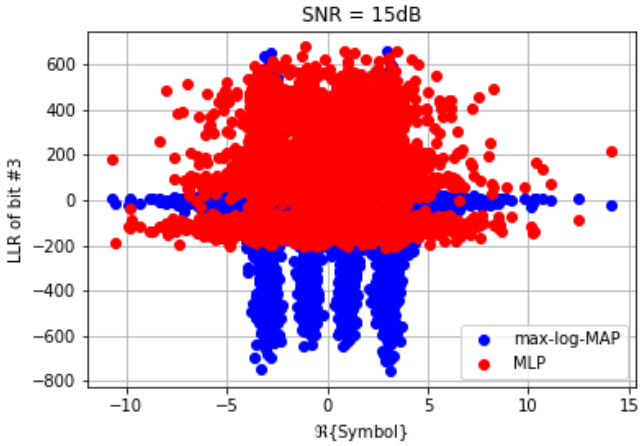

SNR = 20dB
LLR of bit #1
ℜ{Symbol}
max-log-MAP
MLP

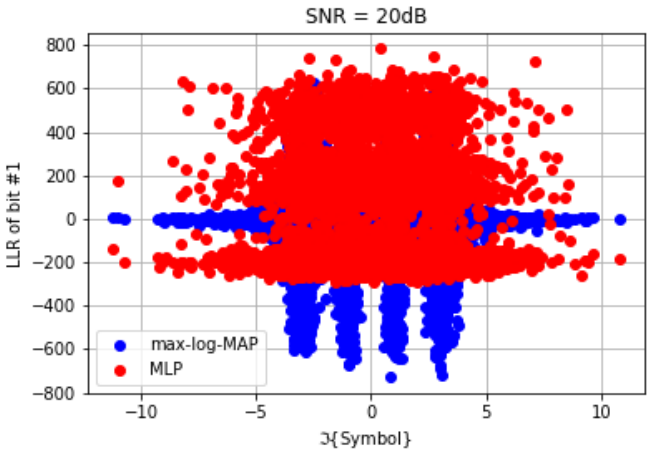
16QAM

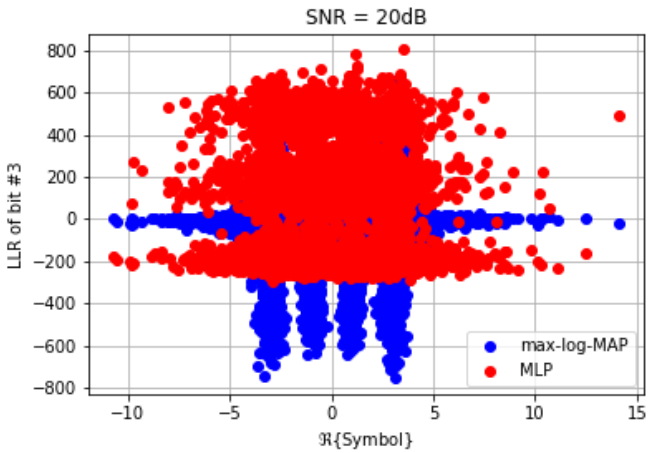

SNR = 10dB
max-log-MAP
MLP
LLR of bit #1
ℑ{Symbol}


SNR = 10dB
max-log-MAP
MLP
LLR of bit #3
ℜ{Symbol}


SNR = 15dB
max-log-MAP
MLP
LLR of bit #1
ℑ{Symbol}


SNR = 15dB
max-log-MAP
MLP
LLR of bit #3
ℜ{Symbol}


SNR = 20dB
max-log-MAP
MLP
LLR of bit #1
ℑ{Symbol}


SNR = 20dB
max-log-MAP
MLP
LLR of bit #3
ℜ{Symbol}

V. CONCLUSION

The objective of this project is to use a neural network to try and predict Log Likelihood Ratios for transmitted bits in order to have a more efficient and cost-effective method of classifying the bits transmitted. The channel was simulated using five different modulation methods, which included 4QAM, 16QAM, 64QAM, BPSK, and 8PSK. The log likelihood ratios were then calculated and used to train the neural network. A Multilayer Perceptron neural network is trained to demodulate symbols to their bit LLRs, and another network is used to simulate a Rayleigh Fading Channel to predict outcomes based on a symbol for a given modulation. The neural networks can be further improved to more accurately predict LLRs by experimenting with more hidden layers and larger training sets. In the future, this technique may have a profound influence in the design of receivers as using a neural network to predict LLRs is much more cost effective than direct computation, whose complexity increases with the order of modulation. In order to improve the bit error rate, a linear block coding technique was used to encode the bits using the Hamming method before converting them to symbols. The results showed that this method did not improve the BER, probably due to the low capacity of MATLAB to carry out a large simulation. Furthermore, a Rayleigh Fading Channel was used instead of a pure AWGN channel in order to have a more realistic transmission when it comes to transmitting radio signals or if we want to have a good model for wireless transmission through urban areas. The BER for the Rayleigh Fading Channel was much higher than the AWGN channel, in general. Overall, the project was very successful in using a neural network to predict LLRs, which is probably going to be very beneficial in the future as machine learning becomes more incorporated into existing systems.